분산 스페어 디스크를 이용한 RAID 패리티 디클러스터링 방법

백운천, 장태무 {wbeak, jtm}@cakra.dongguk.ac.kr
동국대학교 컴퓨터공학과

RAID Parity Declustering using Distributed sparing

Wunchon Beak, Taemu Chang
Dept. of Computer Engineering, Dongguk University

요 약

RAID는 고 병렬성과 고 가용성을 목표로 제안된 대용량 저장 수단이다. 패리티 디클러스터링을 이용한 RAID는 특히 고장이 발생한 경우에도 성능의 저하를 최소화하여 고 가용성을 갖는 저장 장치를 구축할 수 있는 기법이다. 본 논문에서는 이러한 패리티 디클러스터링을 사용한 RAID에 스페어 유닛을 분산시킨 구성을 제안하고, 특히 이러한 분산 스페어링이 고장이 없는 정상 상태에서도 성능 개선에 유용함을 보인다. 본 논문의 실험 결과는 시뮬레이션 방식으로 입증하였으며, 전반적으로 정상상태의 성능을 5-15% 정도 높일 수 있음을 알 수 있다.

1. 개 요

RAID(Redundant Arrays of Inexpensive Disks)는 1980년대 이후 대표적인 대용량 저장 수단으로 많은 응용 분야에서 활용되고 있으며, 시장 규모도 급신장하고 있다[1,2]. 반도체 기술의 급속한 발전으로 마이크로프로세서의 성능은 매년 25-30% 정도 증가하고 있으나, 기계적인 부품을 가진 자기 디스크의 성능 향상 속도는 5%에 불과하다[3]. 따라서 입출력 시스템이 상대적으로 느리기 때문에 발생하는 전체 컴퓨터 시스템의 성능 저하를 막기 위하여 병렬성과 고 가용성을 가진 RAID의 중요도는 날로 크게 부각되고 있다.

RAID 시스템의 구현 방법은 신뢰도를 높이기 위한 패리티 등의 추가정보(redundancy)의 배치 방법에 따라 여러 가지가 있다[4]. 흔히 RAID 단계(level)라고 부르는 것이 그것으로 단계1에서 6까지가 많이 사용되고 있으며, 성능, 신뢰도, 그리고 구축비용 면에서 차이가 있다.

패리티 디클러스터링(parity declustering) 방법은 RAID의 가용성을 높이는 방안의 하나로 많이 연구된 배치(layout) 방법이다. 즉 디스크 배열 중 한 디스크에 고장이 발생하였을 때 및 그 디스크에 있던 자료를 다시 만들어 안전한 디스크로 옮기는 재구축(reconstruction) 작업이 진행 중일 때도 RAID 5 보다 우수한 것으로 알려져 있다.[1, 3, 5, 6, 7]. 반면 디클러스터링 방법은 디스크 배치를 위하여 표를 기억하기 위한 기억 장소 및 위치를 찾기 위한 시간적인 오버헤드가 필요하다. 또한 고장 상태가 아닌 정상 상태(normal mode)에서는 RAID 5보다 성능이 떨어 질 수도 있다.

RAID에서는 하나 이상의 디스크 고장을 대비하여 재구축 작업을 수행하기 위한 스페어 디스크를 두게 된다. 스페어 디스크의 구축 방법은 독자적인 디스크를 두거나, 스페어 부분을 여러 디스크에 분산시키거나, 패리티를 이중으로 두는 방법 등이 제시되어 있다[8]. 디스크 고장 발생 후 재구축 작업이 진행 중일 때 분산 스페어링 방법을 이용하게 되면 부하가 분산되므로 재구축 작업 시간을 단축할 수 있다[1, 3]. 물론 이 경우 두 번째 고장이 일어날 것을 대비하여 스페어 부분에 기록된 자료를 안전한 장소로 대피시키는 오버헤드가 따르지만 디스크 유휴 시간을 이용한다면 무시할 수 있다[3].

본 논문에서는 디클러스터링 방법의 배치를 갖는 디스크 배열에서 분산 스페어링을 도입하는 방안을 제시한다. 기존의 연구[3]에서 이러한 방법이 제안되었으나, 본 논문에서는 개선된 분산 스페어링 방안을 제시하고, 또 분산된 스페어 부분을 정상 상태에서도 활용하여 디클러스터링의 단점 중의 하나인 정상 상태에서 성능이 떨어질 수 있는 점을 보완하고자 하였다.

2. 패리티 디클러스터링의 특성

패리티 디클러스터링은 기본적으로 재구축 비용을 줄이기 위하여 패리티 스트라입(stripe)의 크기(G)와 디스크 배열의 크기(C)를 다르게 하는 배치 방법이다. 이때 $(G-1)/(C-1)$를 디클러스터링 비율(α)이라고 정의한다.

Offset	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	D0,0	D0,1	D0,2	P0	P1
1	D1,0	D1,1	D1,2	D2,2	P2
2	D2,0	D2,1	D3,1	D3,2	P3
3	D3,0	D4,0	D4,1	D4,2	P4

(그림 1) 디클러스터링의 예

(그림 1)은 G=4, C=5, α=0.75인 경우의 예이다. (그림 1)에서 Dn,m은 n번째 스트라입의 m번째 유닛(unit)을 의미하고, Pn은 n번째 스트라입에 대한 패리티이다.

패리티 디클러스터링 배치 구조는 효율적인 자료의 배치라는 새로운 문제를 야기한다. 즉, 단일 고장에 대하여는 올바르게 고치는 것이 가능하도록 한 스트라입의 어떠한 두 유닛도 같은 디스크에 배정하지 않아야 하며, 쓰기 동작의 성능을 높이기 위하여 전체적으로 패리티가 모든 디스크에 분산되어야한다. 또 재구축 부하를 균등하게 하려면 임의의 디스크 쌍에 대하여 동일한 개수의 스트라입이 양 디스크에 유닛을 두어야 하고, 시간 및 공간적으로 효율적인 디스크 주소로 매핑(mapping) 될 수 있는 배치 계획이 필요하다[3, 5]. 그외에도 큰 크기의 쓰기 동작과 읽기 동작을 빠르게 할 수 있는 배치가 좋다.

그러나 위의 여섯 가지 조건을 모두 만족하는 이상적인 배치는 존재하지 않는다. 일반적으로 패리티 디클러스터링 배치를 위하여 사용하는 방법은 블록 설계(block design) 방법론이다. 즉 v 개의 구별되는 개체를 k 개의 원소를 가진 b 개의 튜플(tuple)로 만들되, 각 개체는 정확히 r 개의 튜플에 나타나게 하고, 각 개체의 쌍은 정확히 λ_p 개의 튜플에 나타나게 해야 하며, b는 최소치가 되어야 한다[7].

예를 들어 b=5, v=5, k=4, r=4 그리고 λ_p=3 인 경우 블록 설계의 결과 다음과 같이 튜플들이 만들어진다.

튜플 0 : 0, 1, 2, 3　　튜플 1 : 0, 1, 2, 4
튜플 2 : 0, 1, 3, 4　　튜플 3 : 0, 2, 3, 4
튜플 4 : 1, 2, 3, 4

이를 기반으로 만든 배치가 (그림 1)이다.

(그림 1)의 배치를 만든, 총 $\binom{C}{G}$개의 튜플을 만들어 내는 방법을 완전 블록 설계(complete block design)라고 한다. 그러나 대규모의 배열에서는 튜플 테이블을 기억하기가 불가능하므로 그 중 일부만 사용하는 BIBD(Balanced Incomplete Block Design) 방법을 이용한다[3, 5, 6, 7].

또한 (그림 1)에서 알 수 있듯이 패리티는 디스크 5에 몰려있어서 패리티 분산이 이루어져있지 않다. 실제의 디클러스터링 방법에서는 블록 설계에서 만들어진 튜플을 반복시키되 패리티를 두는 순서를 달리하여 디스크 배치를 행한다. (그림 1)은 전체 블록 설계 테이블(full block design table)의 일부에 해당하며, (그림 1)의 테이블을 k개만큼 반복시킨다.

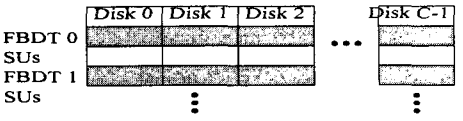
패리티 디클러스터링의 이러한 특성에서 볼 때, 패리티가 완전히 균등하게 분포되지 않으며, 전체적으로 균등하다 하더라도 가까운 거리 내에서는 균등할 수 없다. 이러한 면에서 일반적인 RAID 5보다 정상 상태에서 성능이 떨어 질 수 있는 요인이 있다.

3. 분산 스페어링을 이용한 디클러스터링

일반적으로 독자적인 디스크를 사용하는 스페어링을 RAID 5에 적용하였을 때 발생할 수 있는 문제점은 다음과 같다. 우선 독자적인 디스크는 정상 동작일 때 쉬고 있으므로 자원의 활용도가 떨어지며, 병렬성이 낮아진다. 또한 재구축 작업 시 독자적인 스페어 디스크에 과도한 부하가 걸리므로 재구축 작업 시간 자체도 길어진다. 재구축 작업이 진행될 때는 사용자 요청을 처리하는 시간이 길어 질 수밖에 없으므로 입출력이 빈번한 응용에서는 디클러스터링의 장점을 저해할 수 있다. 특히 낮은 α 값을 갖는 디클러스터링 구조에서는 더 큰 문제가 될 수 있다[3, 8].

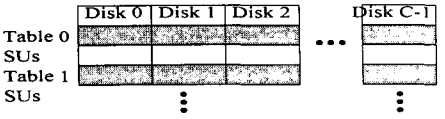
분산 스페어링을 패리티 디클러스터링에 적용하게 되면 위의 문제를 완화할 수 있다. 즉 스페어 유닛을 적절하게 디스크에 할당(allocation)하고, 한 디스크에 고장이 나면 고장이 난 부분의 자료와 패리티를 복원하여 스페어 유닛으로 지정(assignment)하여 재구성(reconfiguration)해야 한다. 다만 스페어 부분을 여러 디스크에 할당할 때 각 디스크에 골고루 분산해야 하며, 재구성 후에도 단일 고장을 감내할 수 있도록 되어야 하며, 재구성 후에도 패리티가 균등하게 분포되는 것이 좋다. 또한 디스크 공간의 손실이 없어야하며, 스페어 유닛이 고장난 부분에 가까운 것이 바람직하다[3].

본 논문에서 고려한 분산 스페어링 방법은 (그림2)와 (그림3)으로 나타내었다.



FBDT=Full Block Design Table
SU=Spare Unit

(그림 2) 분산 스페어링 방법 1



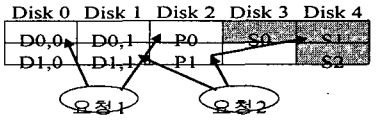
(그림 3) 분산스페어링 방법 2

(그림2)는 [3]과 유사한 방법이며, (그림3)은 스페어 부분을 자료와 패리티 부분에 더 가깝게 둔 방법이다.

본 논문에서는 이러한 스페어 부분을 이용하여 2장에서 이야기된 부분적인 패리티의 불균형 문제를 완화시켜 정상 상태에서의 성능을 높이는 방안을 고려한다. 즉 현재 사용 중인 디스크 내의 패리티에 접근해야하는 요청이 있을 때 이를 사

용되고 있지 않는 디스크의 스페어 유닛으로 우선 접근하여 해결하고, 나중에 제대로 복원하는 방법이다.

예를 들어 (그림4)에서 Disk2에 대한 두 개의 동시에 진행될 수 없는 요청이 있는 경우 요청2의 패리티 쓰기 요청은 S1에서 이루어지게 한다. 따라서 2개의 요청은 동시에 처리할 수 있다. 이 경우 S1 부분에 기록된 정보는 디스크 유휴 시간에 다시 P1 부분에 복사된다.



(그림 4) 상충되는 요청의 처리

4. 성능 분석

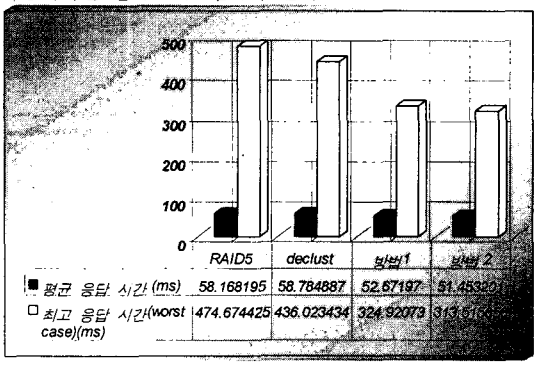
본 논문에서는 3장에서 이야기한 스페어 유닛을 분산시켜 정상 상태에서 성능이 개선됨을 보이기 위하여 시뮬레이션 방법을 사용하였다. 사용한 시뮬레이터는 DiskSim[9, 10]을 본 논문의 디스크 구성에 맞도록 수정한 것이다. DiskSim은 실행 시간 면에서 효율적이고, 다양한 입출력 장치 환경에 사용할 수 있으며, 특히 디스크 장치 단계에서 정확한 시뮬레이션이 가능하다고 알려져 있다. 작업 부하는 DiskSim의 내부 인공 트레이스 합성 기능을 이용하였으며, 이는 UNIX 다수 사용자 환경에서 대형 소프트웨어를 여러 명의 사용자가 수정 및 디버그 작업을 하는 동작을 모방한 것이다[10].

본 논문에서 사용한 시뮬레이션 변수는 (표1)과 같다.

(표 1) 시뮬레이션 변수

~~디스크 배열~~ 스트라입 유닛 : 8KB 패리티 디클러스터링 및 분산 스페어링 디스크의 수 : SCSI 버스로 최대 10개까지 연결
~~디스크(HPC-3323A)~~ 용량 : 1.03 GB 실린더 2982, 표면 7, 섹터 크기 512 B, 8 존(zone) 평균 탐색시간: 10 ms 헤드 회전 속도: 5400 rpm

시뮬레이션 결과는 (그림5)와 같다.



(그림 5) 시뮬레이션 결과

(그림 5)에서 RAID5, α=0.75인 경우 분산 스페어링을 사용하지 않는 디클러스터링, 분산 스페어링을 사용하는 방법1 및 방법2에 대한 평균 응답시간 및 최악의 경우 응답 시간을 보였다. 평균적으로 평균 응답 시간 면에서 약 10% 정도의 성능 향상을 보임을 알 수 있다. 방법1과 방법2의 차이는 크지 않았다.

5. 결론

본 논문은 디스크의 가용성을 높이는 것이 목적인 패리티 디클러스터링 구조에 스페어 유닛들을 분산 시켜 단순히 고장이 발생하여 재구축 작업이 진행 중일 때 사용자 요청을 신속하게 처리해 줄 수 있을 뿐 아니라 재구축 작업을 빠르게 하고, 정상 상태에서도 성능이 개선될 수 있도록 하였다. DiskSim을 수정한 시뮬레이터를 이용한 시뮬레이션 결과 단순한 디클러스터링과 비교하였을 때 5-15%정도의 성능 향상을 기대할 수 있다.

참고 문헌

[1] P. M. Chen et al, RAID: High-Performance, Reliable Secondary Storage. ACM Computing Survey, vol.26, no.2, June 1994, pp.145-185.

[2] G. A. Gibson, Tutorial on Storage Technology: RAID and Beyond, Proceeding of the ACM SIGMOD International Conference on Management of Data, 1995.

[3] M. C. Holland, On-Line Data Reconstruction in Redundant Disk Arrays, PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1994.

[4] D. Patterson et al, A case for Redundant Arrays of Inexpensive Disks, Proceeding of the ACM SIGMOD International Conference on Management of Data, pp.109-116, 1988.

[5] G. A. Alvarez et al, Declustered Disk Array Architecture with Optimal and Neal-optimal Parallelism, The 25th Proceedings of International Symposium on Computer Architecture, 1998, pp.109-120.

[6] E. J. Schwabe et al, Improved Parity Declustering Layouts for Disk Arrays, Proceeding of Symposium on Parallel Algorithms and Architectures, pp.76-84, 1994.

[7] M. Holland et al, Parity Declustering for Continuous Operation in Redundant Disk Arrays, Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, pp.23-35, 1992

[8] J. Menon et al, Comparison of Sparing Alternatives for Disk Arrays, The 19th International Symposium on Computer Architecture, 1992, pp.318-329

[9] G. R. Ganger et al, The DiskSim Simulation Environment Version 1.0 Reference Manual, Technical Report CSE-TR-358-98, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1998.

[10] G. R. Ganger, System-Oriented Evaluation of I/O Subsystem Performance, CSE-TR-243-95, Department of EECS, University of Michigan, Ann Harbor, June, 1995.